Checker Design for On-line Testing of Xilinx FPGA Communication Protocols

Martin Straka, Jiri Tobola, Zdenek Kotasek
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, Brno, 612 66, Czech Republic
strakam@fit.vutbr.cz, kotasek@fit.vutbr.cz
xtobol01@stud.fit.vutbr.cz

Abstract

In the paper, a methodology of developing checkers for communication protocol testing is presented. It was used to develop checker to test IP cores communication protocol implemented in Xilinx FPGA based designs. A formal language enabling to describe the protocol was created for this purpose together with a generator of the formal description into VHDL code. The VHDL code can be then used for the synthesis of the checker structure and used in applications with Xilinx FPGAs.

1 Introduction

With the lower hardware reliability possibly appearing in future technologies, concurrent on-line testing becomes a strong feature in the design of fault-tolerant systems. Concurrent on-line testing implies that testing occurs when the circuit-under-test (CUT) is running in its functional operation mode. One of the ways how to provide on-line testing is through comparing output values produced by CUT with reference values produced by identical circuit operating under the same input vectors, or by a logic block synthesized to produce the same output values as the CUT. In [8], it is demonstrated how checking functions can be used to increase fault coverage and reduce fault detection times. The problem of fault latency is discussed as well. The fault latency is seen as the time it takes to detect a fault after it occurs. In some applications it is important to provide on-line checking of a certain type of communication, e. g. asynchronous handshakes [9].

As mentioned above, fault-tolerant systems design is one of the areas where on-line testing can be possibly used. Fault-tolerance is an important system metric for many operating environments. A possible technique for improving system reliability is through component replication, which usually comes at significant cost: increased design time, testing, power consumption, volume, and weight. Reconfigurable systems implemented using user-programmable logic elements such as FPGA are well suited for applications where high dependability is required [5]. For these applications the problem of radiation effects on SRAM-based FPGAs must be studied [2]. The problems combined with the design of dependable systems include error detection during system operation, fast fault location, quick recovery from temporary failures, and fast permanent-fault repair.

In [3] the method of highly reliable digital circuit design method based on totally self checking blocks implemented in FPGAs is described. The bases of the self checking blocks are parity predictors. The parity predictor design method based on multiple parity groups is proposed. Proper

parity groups are chosen in order to obtain minimal area overhead and to decrease the number of undetectable faults.

The features of fault-tolerance can be implemented in different levels and applications. In [10], a fault-tolerant approach to reliable microprocessor design is proposed. The approach, based on the use of an on-line checker component in the processor pipeline, provides significant resistance to core processor design errors and operational faults such as supply voltage noise and energetic particle strikes [10].

The problem of on-line testing is widely discussed in numerous papers. In [4], it is presented how path (min) delay faults when designing on-line testable circuits should be considered. The challenges that this poses to the existing on-line testing strategies are discussed. Examples showing the possible incorrect behaviour of a self-checking circuit as a result of this kind of faults are given. In [1], the idea of combining self-test technology for production test and for online self test is presented. The reduction of overall overhead for testing is the goal of the activity.

2 Motivation for the Research

Hardware units can be implemented on various platforms. From among those which are widely used in many applications, Xilinx FPGA can be mentioned. For the purpose of interconnecting FPGA components, Xilinx company developed a Local Link (LL) protocol which has been integrated to many IP cores.

Very often it is reported that FPGA based designs are constructed as fault tolerant designs with the possibility of recovering from errors by means of reconfiguration procedures. In our opinion, testing proper function of communication protocol can increase significantly the diagnostic quality of the design. Therefore, we have decided to develop a methodology for automatic design of Local Link communication protocol checker. In the first phase of our research we decided to develop a checker which will operate on different levels of detecting communication protocol faults: 1.a checker detecting an incorrect combination of output signals 2. a checker detecting a correct sequence of signals 3. a checker constructed as a FSM whose state reflects the combination of signals.

The complexity of the checker will be different based on the type of communication protocol fault supposed to be detected by the checker. The complexity of the checker will influence the area required on the chip and communication speed. As an important aspect of the methodology we saw that the alternative of automated design of the checker should be available to a designer. For this purpose, we felt the need for a language by means of which the conditions supposed to be checked will be described together with the need for core generator to compile checker description into VHDL code.

The paper is organized in the following way: in Section 3 the definition of the language for the description of possible communication protocol faults is given. Section 4 devotes to the core generator which was developed for the automated checker design from the definition language, while the results are evaluated in Section 5. In Section 6 the results are recapitulated and in Section 7 our goals for future research in this area are summarized.

3 Language Definition

If a digital system is expected to be fault-tolerant, then not only an erroneous operation of all components must be guaranteed but also the correctness of communication protocols becomes a

strong aspects of the system reliability. Thus the need for the development of automatic checkers of communication protocols must be reflected during fault-tolerant system design. Usually, to describe errors in communication protocols, formal models such as grammars, FSMs, or formal languages are used. As a result of our research a language was developed which allows to describe possible failures in communication protocol. The description is then used as an input to automatic generator which develops checker description in VHDL language.The main advantage of this approach is such that based on the language the checker can be generated automatically without the intervention of experienced designer.

When a communication protocol is checked, then not only the combinations of signals must be monitored but also their sequences. The checker behavior must therefore have features of sequential behavior which can be described by means of FSM. The definition of language for communication protocol errors detection therefore arises from the formal description of FSM.

The language description is composed of two parts. The first one defines the input alphabet symbols that uniquely specify the transitions between automata states. Each input symbol is defined as the set of conditions over the communication protocol signals. The second part of the language defines the transition function of automata. For each state and input symbol, the transition to the next state is defined, as show in Figure 1.

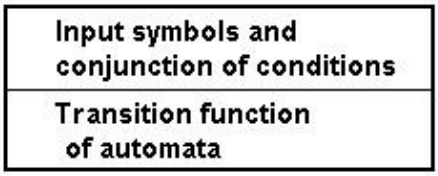

Figure 1. Phases of core generator processing

The initial state is labeled as *S0*, it must be as the initial state in the transition function of automata. The error during the communication protocol is detected by the transition to *Serr* state. If the automata is not completely defined, then it remains in the previous state for uncovered input signal combinations.

The input automata symbols are defined as conjunction of conditions $cond_1$, $cond_2$, ..., $cond_N$ separated by $and, or, (,)$ symbols and ended by $semicolon$. Each condition contains single comparison operators ($<$, $>$, $<=$, $>=$, $==$ and $<>$) between signal and numeric constant. Syntactic structure is shown in the following example:

The conditions are defined in the following way:

$$cond0 \rightarrow Name\_signal == 1$$
$$cond1 \rightarrow Name\_signal >= 0$$

Expression defined in such manner is assigned to new input symbol.

$$p0 = cond0 \text{ and } cond1 \text{ or } (cond2 \text{ or } cond3) \text{ and } ... \text{ and } condn;$$

The automata behaviour is described using transition function which is represented by a set of transitions in the form:

$$(S0, p0) : S1;$$
$$(Serr) : S0;$$

Based on the set of input characters and transition function, it is possible to construct formally the finite state machine and design the checker. An example of the simple checker behavioral description is shown here:

$$p0 = sigA == 1 \text{ and } sigB <> 0 \text{ and ... and } sigC <> 0;$$
$$p1 = (sigC >= 0 \text{ or } sigA < 1) \text{ and ... and } sigD == 0;$$
$$(S0, p0) : S1; (S1, p1) : S3;$$
$$(S3, p0) : S0; (S3, p1) : Serr;$$
$$(Serr) : S0;$$

$$A = (Q, T, P, S0, Serr)$$

where $Sn \in Q$ is the set of the all states, $Pn \in T$ is the set of input symbols, P is the function which for each state and input symbol defines the following state. $S0 \in Q$ is the starting state and $Serr \in Q$ is the error state of the checker.

The proposed language is able to describe any automata that reacts to communication protocol signals. Other operators extend the ability of checker to detect more types of errors at the functional level. For example, it is possible to detect, if the appropriate input values fall into the required ranges.

4 Core Generator

Core generator is a program for automated development of checker structure based on the description provided in formal language. By means of the formal language the conditions of communication protocol are described. The process of generating checker consists of two phases, Figure 2:

1. PHASE: The input file is analyzed, the conditions which must be satisfied together with transition functions are transformed into FSM description.
2. PHASE: The transitions reflected by FSM description are mapped into VHDL processes.

As the first step of the input file analysis, the symbols of the files are analyzed together with conditions assigned to them. The set containing all input symbols is created and the syntax analysis of conditional statements is performed. For each conditional statement a syntax tree is formed which is then used during mapping the conditions onto the description in VHDL language.

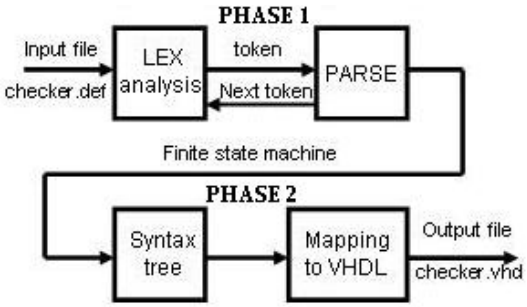

Figure 2. Phasis of core generator processing

As the result of the analysis, an FSM is constructed, A = (Q,T,P,S0,Serr), where Q is the set of all states, T is the set of input symbols, P is the set of transition conditions, S0 is the initial state and the Serr is the error state indicating error in communication.

The second phase starts with creating the interface of the checker. The names of signals are extracted from transition conditions. The conditions are then mapped onto VHDL processes. The interface signals are the input to the process, the output of the process is the only signal, whose name reflects one of input symbols. The contents of the process is generated from the syntax tree developed in the first phase of the analysis. The mapping of FSM into VHDL is performed by means of two processes. One of them operates as a register in which current state is stored and the second process describes the combinational logic reflecting transition conditions.

5 Evaluation of the Methodology

The proposed approach for generating checker structure was tested on Local Link communication protocol [11] developed by *Xilinx* company which is used especially for FPGA components interconnection. The Local Link protocol has been integrated to many IP Cores. As an example of the most familiar IP Cores the following designs can be used: Aurora that provide communication through RocketIO multigigabit transceivers; PCI Express IP Core for communication with PCI interface; GMII/XGMII Cores for packet receiving and transmitting via 1Gbps or 10Gbps Ethernet and many others. Unfortunately, most of IP Cores do not offer any diagnostics opportunities such as BIST or other on-line/off-line test. In this case, the solution in the form of hardware component checking the communication protocol and transferred data can detect the most critical faults.

The Local Link (LL) is based on synchronous point-to-point communication protocol which transfers data in the form of packets. To the LL advantages generic data width of transferred data belongs which is a very important aspect for stream processing applications. Additionally, LL offers upstream and downstream flow control, efficient link bandwidth utilization and optional parity checking. The LL interface contains 6 control signals, data bus and signals identifying the number of valid bytes available in the last data word (REM).

Two control signals (SRC_RDY_N and DST_RDY_N) participate in the flow control, allowing both communication sides (source and destination component) can stop the communication. Other four control signals are used for identifying the structure of transferred packet. SOF_N specifies the start of frame, SOP_N identifies the end of the header and the beginning of packet payload, EOP_N determines the end of the payload and the start of the footer. Finally, EOF_N specifies the end of the frame. All control signals are active in L level. The example of Local Link communication protocol is shows in Figure 3. Detailed specification of Local Link protocol is available in [11].

For the purposes of checker evaluation, three different levels of diagnosis were chosen:

(1) As the simplest checker alternative we see the monitoring of control signals and the detection of correct combinations on the protocol interface, the current status of the communication need not be identified.

(2) The second level verifies the correct sequences of control signals and evaluates the transitions between communication protocol states.

(3) At the third level, the content of data is verified whether. It is checked whether specified conditions and rules are satisfied.

From LL protocol specification the following correct signal combinations (1) can be derived, SRC_RDY_N and DST_RDY_N being active:

1. Every frame must start with SOF_N signal and no other signal is allowed to be active.

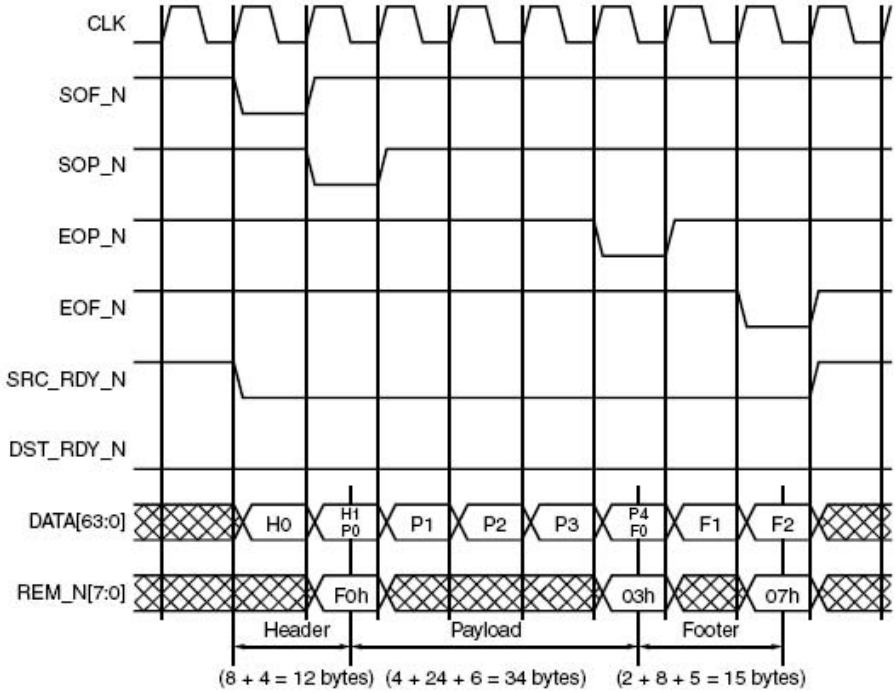

Figure 3. Local Link Protocol Timing Diagram

2. Each frame must contain a header at the beginning. Thus, if SOP_N is active, no other signal is allowed to be active.
3. Each frame must contain a footer. Thus, if EOP_N is active, no other signal is allowed to be active.
4. Each frame must be accomplished with EOF_N signal, no other signal is allowed to be active.
5. If data is transported, no other signal except of SRC_RDY_ N and DSC_RDY_N signals is allowed to be active.

This list of rules can be easily rewritten into the language (defined in section 3) as follows:

$$
\begin{aligned}
p0 &= SRC\_RDY\_N == 0 \; and \; DST\_RDY\_N == 0 \; and \; SOF\_N == 0 \\
&\quad and \; SO\_N == 1 \; and \; EOP\_N == 1 \; and \; EOF\_N == 1 \\
p1 &= SRC\_RDY\_N == 0 \; and \; DST\_RDY\_N == 0 \; and \; SOF\_N == 1 \\
&\quad and \; SO\_N == 0 \; and \; EOP\_N == 1 \; and \; EOF\_N == 1 \\
p2 &= SRC\_RDY\_N == 0 \; and \; DST\_RDY\_N == 0 \; and \; SOF\_N == 1 \\
&\quad and \; SO\_N == 1 \; and \; EOP\_N == 0 \; and \; EOF\_N == 1 \\
p3 &= SRC\_RDY\_N == 0 \; and \; DST\_RDY\_N == 0 \; and \; SOF\_N == 1 \\
&\quad and \; SO\_N == 1 \; and \; EOP\_N == 1 \; and \; EOF\_N == 0 \\
p4 &= SRC\_RDY\_N == 0 \; and \; DST\_RDY\_N == 0 \; and \; SOF\_N == 1 \\
&\quad and \; SO\_N == 1 \; and \; EOP\_N == 1 \; and \; EOF\_N == 1 \\
p5 &= SRC\_RDY\_N == 0 \; or \; DST\_RDY\_N == 0
\end{aligned}
$$

This approach is limited and can detect only the basic faults caused by forbidden combinations of signals in the protocol interface.

The second type of rules considers sequences of control signals. For the Local Link protocol the following transition rules can be applied:

$$(S0, p5) : S0; (S0, p0) : S1;$$
$$(S1, p5) : S1; (S1, p1) : S2; (S1, p4) : S1;$$
$$(S2, p5) : S2; (S2, p2) : S3; (S2, p4) : S2;$$
$$(S3, p5) : S3; (S3, p3) : S0; (S3, p4) : S3;$$
$$A = (Q, T, P, S0, Serr)$$

The last part covers data monitoring transported by means of the protocol, checking the conditions rules describing the contents of data. An example: the first transported byte must contain 0xAB (Start-of-Frame Delimiter), the ninth byte must have the value which is lower then 124 (the width of the word is 4 bytes). The record in the language is given by the following rules:

$$p0 = DATA_0[7\ downto\ 0] == 0xAB;$$
$$p1 = DATA_2[7\ downto\ 0] < 124;$$

$$(S0, p0) : S1;$$
$$(S1, p1) : S0;$$

6 Experimental Results

For all types of rules checker structure was generated and correct behavior was tested on COMBO6X card with FPGA Virtex2 Pro for network traffic [7]. Synthesis to Virtex2 Pro FPGA was also performed to obtain basic parameters of generated circuit. For all generated circuits, the maximal frequency was higher than 350 MHz and does not affect maximal frequency of IP cores. FPGA logic utilization was different for all types of rules and types of diagnostic levels. The results show that the amount of used resources is really low. All results are summarized in table 1.

The types of rules were derived from the LL protocol specification developed for FPGA VIRTEX2Pro. For each of three types of rules for which checkers were synthesized, different numbers of slices were involved into the design as the result of the synthesis. The right column of the table gives the numbers of slices needed to detect error states of the protocol. In both cases, power requirements are very low. In our opinion, in practical applications it will be reasonable to monitor correct combinations and sequences of protocol signals.

Rules	Slices	Rules	Slices
Correct Signal Combination	3	Error Signal Combination	4
Correct Sequence of Signals	5	Error Sequence of Signals	7
Data checking	16	Data checking	16

Table 1. Resources usage in slices

For the third type of rules, where also data is checked, the number of utilized resources is related to the number of compared bits. The results of our experiments show that the number of utilized resources is increased linearly with the number of compared bits.

An example of checker constructed for Local Link protocol is shown in Figure 4.

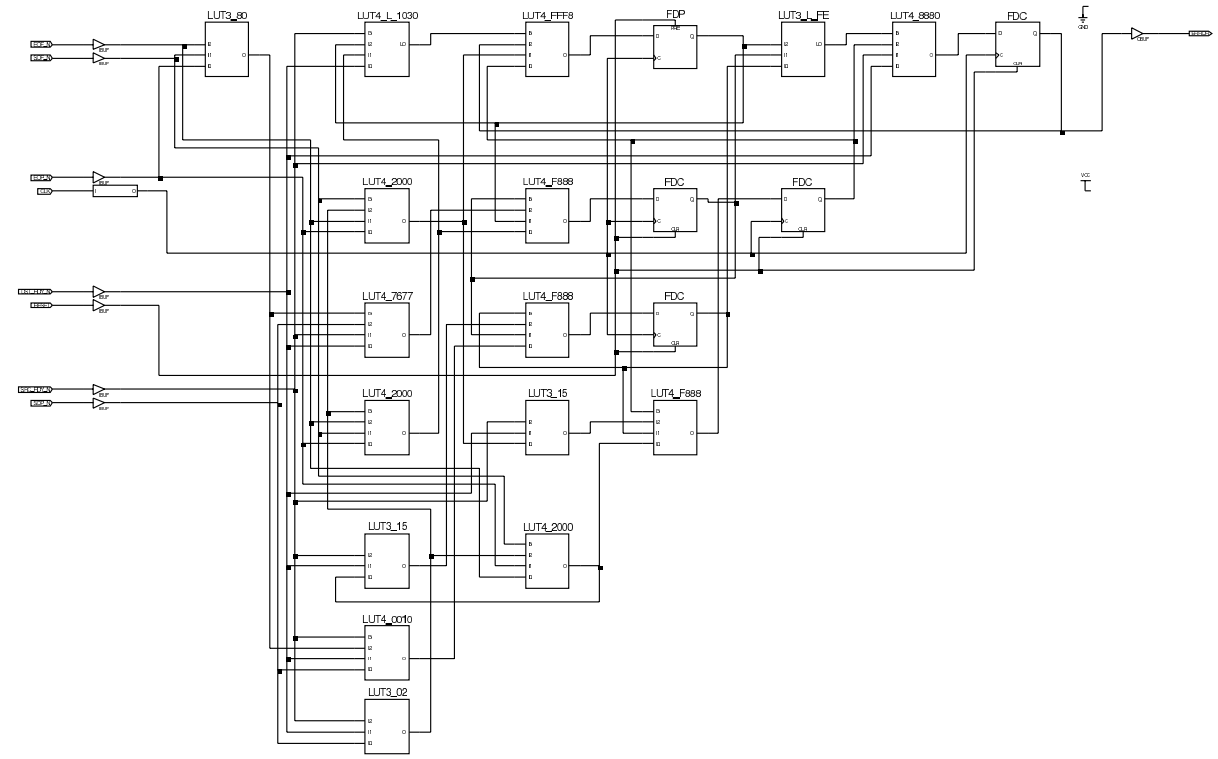

Figure 4. Checker implementation for Local Link protocol

7 Conclusions

A methodology was developed which can be used for automatic development of communication protocol checkers. The following tasks had to be solved during the work on the topic: 1. the development of formal tool for the formal description of properties that are supposed to be checked - the properties are derived from protocol definitions and rules. 2. the implementation of the generator which can be then used for compiling the description into VHDL code, 3. the synthesis of the checker into Virtex 2 FPGA, 4. experimenting with all these tools, comparing the results (in terms of checker complexity) gained for different sets of properties.

Our approach of generating hardware checkers is different from those based on the utilization of PSL [6]. While PSL based methodologies allow to describe formally the properties to be checked and add this description to the HDL source code of the component to be synthesized (and develop the checker together with the component), our methodology can be used in situations where the design is finished and it is still expected that certain properties should be checked. As an example, precisely defined communication protocol can serve for which a checker is needed. In our future research, we have an intension to compare our results with those based on the usage of PSL.

Acknowledgements

This work was supported by the Research Project No. MSM 0021630528 - Security-Oriented Research in Information Technology and by GACR (Grant Agency of the Czech Republic) project No. 102/05/H050 - Integrated Approach to Education of PhD Students in the Area of Parallel and Distributed Systems.

References

[1] C. Galke, M. Grabow, and H. T. Vierhaus. Designing fault-tolerant techniques for sram-based fpgas. *Perspectives of combining on-line and off-line test technology for dependable systems on a chip, IOLTS*, pages 183–188, December 2003.

[2] F. Kastensmidt, G. Neuberger, F. R. Hentschke, L. Carro, and R. Reis. Designing fault-tolerant techniques for sram-based fpgas. *IEEE Design and Test of Computers*, pages 552–562, December 2004.

[3] P. Kubalik, P. Fiser, and H. Kubatova. Fault tolerant system design method based on self-checking circuits. *Proceedings of the 12th IEEE International Symposium on On-Line Testing (IOLTS06)*, pages 185–189, December 2006.

[4] C. Metra, M. Omana, D. Rossi, J. M. Cazeaux, and T. M. Mak. Path (min) delay faults and their impact on self-checking circuits' operation. In *Proceedings of the 12th IEEE International Symposium on On-Line Testing (IOLTS06)*, pages 17–22, Corno, Italy, 2006. IEEE Computer Society.

[5] S. Mitra, W. Huang, and N. R. Saxena. Reconfigurable architecture for autonomous self-repair. pages 228–240, Cannes, France, June 2004.

[6] K. Morin-Allory and D. Borrione. Proven correct monitors from psl specifications. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 1246–1251, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[7] J. Novotný, O. Fučík, and R. Kokotek. Schematics and PCB of COMBO6 card. Technical Report 14/2002, CESNET, 2002. available at http://www.cesnet.cz/doc/techzpravy/2002/combo6/combo6.pdf.

[8] I. Pomeranz and S. M. Reddy. Reducing fault latency in concurrent on-line testing by using checking functions over internal lines. *Proc. of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'04)*, pages 183–190, October 2004.

[9] D. Shang, A. Yakovlev, F. Burns, F. Xia, and A. Bystrov. Low-cost online testing of asynchronous handshakes. *Proceedings of the Eleventh IEEE European Test Symposium (ETS'06)*, pages 225–232, March 2006.

[10] C. Weaver and T. Austin. A fault tolerant approach to microprocessor design. *Dependable Systems and Networks (DSN)*, pages 411–420, July 2001.

[11] Xilinx Inc. 2100 Logic Drive. *LocalLink Interface Specification*. San Jose, September 2006.